INTRUSION DETECTION USING A NETWORK PROCESSOR AND A PARALLEL PATTERN DETECTION ENGINE

CROSS REFERENCE TO RELATED APPLICATIONS

The present invention is related to the following U.S. Patent Applications which are incorporated herein by reference:

Serial No. ____________ (Attorney Docket No. RPS920030020US1) entitled "A Configurable Bi-Directional Bus For Communicating Between Autonomous Units" filed ________; and

Serial No. ____________ (Attorney Docket No. RPS920030036US1) entitled "Parallel Pattern Detection Engine" filed ________.

TECHNICAL FIELD

The present invention relates in general to methods and systems for detecting malicious intrusion in a network of systems.

BACKGROUND INFORMATION

Recognizing patterns within a set of data is important in many fields, including speech recognition, image processing, seismic data, etc. Some image processors collect image data and then pre-process the data to prepare it to be correlated to reference data. Other systems, like speech recognition, are real time where the input data is compared in real time to reference data to recognize patterns. Once the patterns are "recognized" or matched to a reference, the system may output the reference. For example, a speech recognition system may output equivalent text to the processed speech patterns. Other systems, like biological systems, may use similar techniques to determine sequences in molecular strings like DNA.

In some systems, there is a need to find patterns that are imbedded in a continuous data stream. In non-aligned data streams, there are some situations where

patterns may be missed if only a single byte-by-byte comparison is implemented. The situation where patterns may be missed occurs when there is a repeated or nested repeating patterns in the input stream or the pattern to be detected. A reference pattern (RP) containing the sequence that is being searched for is loaded into storage where each element of the sequence has a unique address. An address register is loaded with the address of the first element of the RP that is to be compared with the first element of the input pattern (IP). This address register is called a "pointer." In the general case, a pointer may be loaded with an address that may be either incremented (increased) or decremented (decreased). The value of the element pointed to by the pointer is retrieved and compared with input elements (IEs) that are clocked or loaded into a comparator.

In pattern recognition, it is often desired to compare elements of an IP to many RPs. For example, it may be desired to compare an IP resulting from scanning a finger print (typically 1 Kilobyte for certain combinations of features defined in fingerprint technology) to a library of RPs (all scan results on file). To do the job quickly, elements of each RP may be compared in parallel with elements in the IP. Each RP may have repeating substrings (short patterns) which are smaller patterns embedded within the RP. Since a library of RPs may be quite large, the processing required may be considerable. It would be desirable to have a way of reducing the amount of storage necessary to hold the RPs. If the amount of data used to represent the RPs could be reduced, it may also reduce the time necessary to load and unload the RPs. Parallel processing may also be used where each one of the RPs and the IP are loaded into separate processing units to determine matches.

Other pattern recognition processing in biological systems may require the comparison of an IP to a large number of stored RPs that have substrings that are repeated. Processing in small parallel processing units may be limited by the storage size required for the RPs. Portable, inexpensive processing systems for chemical

analysis, biological analysis, etc., may also be limited by the amount of storage needed to quickly process large numbers of RPs.

Pattern detection or recognition is a bottleneck in many applications today and software solutions cannot achieve the necessary performance. It is desirable to have a hardware solution for matching patterns quickly that is expandable. It is also desirable to have a system that allows multiple modes of pattern matching. Some applications require an exact match of a pattern in an input data stream to a desired target pattern. In other cases, it is desirable to determine the longest match, the maximum number of characters matching, or a "fuzzy" match where various character inclusions or exclusions are needed.

Intrusion Detection Systems (IDSs) provide a means to detect patterns of bytes in packets that are certainly or probably associated with malicious activity. IDSs may operate on host-based systems or on network data flows, called network-based systems. In either case, an IDS looks for attacks (any malicious activity) originating from outside or inside the internal network and acts much like a burglar alarm. This is essentially a pattern recognition task where an IDS analyzes incoming data while attempting to detect known patterns (signatures) that indicate the presence of a known intruder.

Intrusion detection products are tools that assist in the protection of a network from intrusion by expanding the options available to manage the risk from threats and vulnerabilities. Intrusion detection capabilities may help a company secure its information. After an attack is detected by the system, the system may provide information about the attack. This information may be used to delete, log, or shun intruding packets. Support investigations then attempt to find out how the intruder breached the network security and then stop the breach method from being used by future intruders.

Malicious traffic is common in today's Internet. Current IDS performance may become a bottleneck at high bandwidth. Some attackers may launch a high-speed attack in order to overwhelm IDS and simultaneously a low-speed attack in hope that the low-speed attack will not be noticed. These attacks may be real-time, live traffic attacks. This means that computing networks must continually scan traffic to catch these malicious activities. Current IDS software throughput is inversely proportional to the network load, hence is more prone to attacks, and run at higher loads. IDS software enables either comprehensive or high speed detection, but not both.

There is, therefore, a need for a method and circuitry to form an IDS that is able to detect intrusions, identify a variety of attacks, and run at the real-time speed of the high performance network.

SUMMARY OF THE INVENTION

A network processor incorporates a parallel pattern detection engine (PPDE) that contains a large number of processing units (PUs) designed specifically for pattern comparison and detection. The PPDE is highly scalable and does pattern detection in a variety of programmable modes in parallel with each of the PUs receiving input data in parallel. Each of the PUs have received unique identification (ID) data along with the sequence of bytes representing the pattern that it is comparing to the input data stream. Communication between the PUs allows the ID of the PU with desired comparison result data to be outputted on a bus structure coupling all the PUs. When the PUs are loaded with pattern data, global configuration registers are set defining the type of matching to be performed as well as the type of cascading that is desired between adjacent PUs. In this manner, the PPDE uses groups of cascaded PUs to detect larger patterns than are possible with a single PU without resorting to iterative processes which take more time.

The PPDE is an integrated circuit (IC) that has a large number of PUs, may be cascaded with other PPDE ICs, and is configurable via an input/output (IO) bus. The IDS, according to embodiments of the present invention, is formed by coupling one or more PPDE ICs to a network processor (NP). The host computer is used to load known signatures of intruders into the PUs of the IDS as patterns to be matched (detected). As the NP is receiving network traffic (input data), it processes the packets of data and sends the data at network speed to the IDS which compares the input data bytes to pattern data bytes looking for a match. If a match is detected, the ID of the PU detecting the match is outputted which identifies the signature of the intruder. The NP may then execute counter measure code to combat the threat of the detected intruder

The IDS may be configured so that additional PPDE ICs may be added as needed. Since the PPDE ICs are designed so that their PUs receive input data in parallel bus, are programmable, output their comparison results on an output bust and

may be cascaded, the IDS is dynamically configurable without adding additional logic.

The foregoing has outlined rather broadly the features and technical advantages of the present invention in order that the detailed description of the invention that follows may be better understood. Additional features and advantages of the invention will be described hereinafter which form the subject of the claims of the invention.

The foregoing has outlined rather broadly the features and technical advantages of the present invention in order that the detailed description of the invention that follows may be better understood. Additional features and advantages of the invention will be described hereinafter which form the subject of the claims of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, in which:

FIG. 1 is a block diagram of the architecture of a parallel pattern detection engine (PPDE) comprising N processing units (PUs) suitable for practicing embodiments of the present invention;

FIG. 2A-2D are block diagrams of four matching modes which may be programmed for each of the N PUs of FIG. 1;

FIG. 3 is a chart illustrating the various modes of scalability of the PPDE of FIG. 1;

FIG. 4 is a chart of performance results achievable by a PPDE integrated circuit employing 1500 PUs suitable for practicing embodiments of the present invention;

FIG. 5 is an overview block diagram of an individual PU in the PPDE of FIG. 1;

FIG. 6 is a detailed block diagram of an individual PU in the PPDE of FIG. 1;

FIG. 7 is another detailed block diagram of an individual PU in the PPDE of FIG. 1;

FIG. 8 is a circuit diagram of a specific implementation of a single PU in the PPDE of FIG. 1;

FIG. 9 is a flow diagram of method steps used in matching patterns suitable for practicing embodiments of the present invention;

FIG. 10 is the block diagram of a network processor (NP) suitable for practicing embodiments of the present invention;

FIG. 11A-11E illustrate operation in various modes of pattern matching which may be used in embodiments of the present invention;

FIG. 12 is a block diagram of an intrusion detection system (IDS) according to embodiments of the present invention;

FIG. 13 is a flow diagram of method steps used in embodiments of the present invention;

FIG. 14 is a circuit block diagram of communication circuitry for cascading multiple PU 500 units within a PPDE 100 suitable for practicing embodiments of the present invention; and

FIG. 15 is another block diagram of the communication circuitry between a PU 500 and two adjacent PU 500 units suitable for practicing embodiments of the present invention.

DETAILED DESCRIPTION

In the following description, numerous specific details are set forth to provide a thorough understanding of the present invention. However, it will be obvious to those skilled in the art that the present invention may be practiced without such specific details. In other instances, well-known circuits may be shown in block diagram form in order not to obscure the present invention in unnecessary detail. For the most part, details concerning timing, data formats within communication protocols, and the like have been omitted inasmuch as such details are not necessary to obtain a complete understanding of the present invention and are within the skills of persons of ordinary skill in the relevant art.

Refer now to the drawings wherein depicted elements are not necessarily shown to scale and wherein like or similar elements are designated by the same reference numeral through the several views.

Sequential matching of a data stream in software is currently a central processing unit (CPU) intensive task. Thus high performance is difficult. The pattern matching processing unit (hereafter PU) architecture provides high performance matching because it is a piece of hardware dedicated to pattern matching. The PU provides more efficient searching (matching because every input pattern is being matched in parallel to a corresponding target pattern). Parallel matching is possible because a large numbers of the PUs may be cascaded. Additionally, each PU has built in functionality that may reduce the number of necessary PUs by incorporating modes that allow matching comprising wild cards (don't cares in the target pattern), multiple wildcards, and inverse operations. The PU architecture's fast pattern detection capabilities are useful in network intrusion detection, database scanning, and mobile device security applications. Additionally, with their built in distance computation, "fuzzy" pattern detection may be implemented which are particularly useful in image processing and life sciences applications.

FIG. 5 is a overview block of diagram of a PU 500 according to embodiments of the present invention. PU 500 receives inputs from identification (ID) bus 501, control bus 502 and input data bus 503. The inputs of the buses are buffered in ID register 509, control register 505 and input data register 504. Control data from control register 505 is coupled to control logic circuitry 508 which also receives data from memory 507. Input data from input data register 504 is coupled to memory 507, address circuitry 506, masking circuitry 510. Address circuitry 506 couples addresses to memory 507. Address circuitry 506 also couples to masking circuitry 510 and output circuitry 512. Output circuitry 512 receives data from ID register 509, address circuitry 506 and distance circuitry 511 and selectively couples data to output bus 513.

FIG. 6 is another more detailed block diagram of PU 500 according to embodiments of the present invention. Blocks shown in FIG. 5 are repeated for clarity. PU 500 receives inputs from identification (ID) bus 501, control bus 502 and input data bus 503. The inputs of the buses are buffered in ID register 509, control register 505 and input data register 504. Memory 507 is a register array having fields for pattern data 601 and operation codes (Opcodes) 602. Memory 507 stores patterns that are being compared to input data. Opcodes 602 define what type of pattern compare is being executed. Opcodes 602 and control bits from control register 505 are coupled to control logic circuitry 508. Pattern data 601 are coupled to mask register 603 in mask circuitry 510. Outputs of mask register 603 are combined in logic AND 605 to generate inputs to component distance computation unit 610 in distance circuitry 511. Likewise, outputs of mask register 603 are combined in a logic AND 606 to form inputs to data selector 604. Data selector 604 selects between input data from input register 504 and addresses from address register 614 to provide inputs to component distance computation unit 610. Address register 614 couples address to memory 507. Component distance computation unit 610 couples outputs to Pattern distance computation unit 611. Present distance computation results are stored in distance register 612. The present distance computation result is coupled

back to pattern distance computation unit 611 and to compare circuitry 607. The output of distance register 612 is compared to the value stored in the final distance register to generate the output GT 615. GT Stand for "greater than" and this signal is set active when the value stored in the final distance register is greater than the value stored in the distance register. The final distance value in store in final distance register 608 is selected from either input register 504 or distance register 612 in distance selector 609.

Each PU 500 has limited memory to store pattern data 601. If a pattern is long, it is possible to merge several PU 500 units for storing a long sequence of pattern data 601. For example if two PU 500 are used, then during the beginning of a pattern detection phase, the memory 507 of the first of the two PU 500 units is used. The address pointer of the first PU 500 is modified according to the matching mode and the operation codes 602. When the address pointer reaches its last memory position a last signal 650 is sent to the second of the two PU 500 units in order to continue the matching process using the remainder of the pattern data 601 stored in the second PU 500. Control data on control bus 502 is used to initialize the second PU 500, in this case, so that it only starts matching when it receives the "last" signal 650 from the first PU 500. Also in this case, if a "reload" pointer address is indicated during the matching process, the address pointer of both of the two PU 500 units used for the long sequence of pattern data 601 must be updated. This is accomplished by sending a "reload" signal 651 to the appropriate PU 500 (containing the initial pattern 601 bytes). Since the number of bytes in a sequence of pattern data 601 is not specifically limited, more than two PU 500 units may be used in the manner discussed. Again initialization control data on control bus 502 configures a PU 500 to execute as an independent PU or as a cascade PU.

When the matching mode is a "fuzzy" match, pattern distance computation unit 611 calculates a present distance value stored in distance register 612. If two or more PU 500 units are used in cascade to store pattern data 601 used for a fuzzy

match, then the distance value is sent on distance signal 652 to the next PU 500 in a cascade so that a final distance value may be determined and stored in final distance register 608 of the last PU 500 in a cascade.

FIG. 7 is a block diagram of more details of circuitry PU 500. Patterns to be compared are preloaded into memory (register file) 507 as bytes wherein each bit is stored as 8 bits in bits [11:4]. Each Opcode 602 is stored in bits [3:0]. An input data stream 750 are compared to stored bytes in memory 507 as determined by read address 614. Compare and distance unit 511 computes a distance for the compare operation. Match logic 709 generates logic signals that are coupled to reload logic 710, increment logic 711 or hold logic 712. Various types of matching are possible as determined by Opcodes 602 stored with each byte of the pattern in memory 507. Depending on the Opcode 602 and the results of the compare in compare and distance unit 511, the logic in reload logic 710, increment logic 711 and hold logic 712 determine whether to hold the present read address, increment the present read address to the next value or reload the read address to its initial value to start comparing at the beginning of the pattern. Select line logic 705 is enabled by activate logic 713 via activate signal 730. Depending on the output logic states of reload logic 710, increment logic 711 and hold logic 712, one of the inputs to multiplexer (MUX) 704, hold 723, increment 722 or reload 721 will be a logic one thereby selecting input 703, 702 or 701 respectively. Increment by one 714 adds one to the present read address and generates input 702. The present read address is coupled into hold 703 and the first address in the pattern is coupled from 714. Register 614 was loaded with the first address in the pattern under control of Opcodes 602. Packet reset signal 751 resets the read address. If active signal 706 is a logic zero, then select line logic 705 is degated and all the inputs hold 703, increment 702 and reload 701 are a logic zero and MUX 704 is degated. To allow cascading of multiple PUs (e.g., PU 500), the signal 730, and ID 707 are coupled to the next PU. Likewise, PU 500 receives ID 752 and active signal 753 from a preceding PU. Activate logic 713 is coupled to the previous PU by signal line 790.

FIG. 8 is a more detailed circuit diagram of circuitry of PU 500. FIG. 8 illustrates a more detailed circuitry for select line logic 705 (AND gates 760-762), reload logic 710 (OR gate 763 and AND gates 764-765), increment logic 711 (OR gate 766 and AND gates 767-769) and hold logic 712 (AND gate 770). Inverters 780-784 serve to generate the complement of the Opcode 602 signals.

The following description may refer between FIGS. 5, 6, 7, and 8 as these illustrate PU 500 in various degrees of detail.

The fast pattern match technology utilizes local memory (e.g., register array 507) in each PU 500 which contains a pattern 601 and flag bits (Opcodes 602) that specify options. These options may include a single wildcard, multiple wildcard, last, and inverse matching operations. A single wildcard matching means that a match is indicated if the byte having the single wildcard matching Opcode 602 set matches the current byte in an input stream. A multiple wildcard matching means that a match is indicated if an indeterminate number of bytes in sequence not match the byte with the multiple wildcard Opcode 602. Inverse matching means a match is indicated if every byte except the byte with the inverse Opcode 602 matches a byte in an input stream. Last Opcode 602 means that the byte is the last byte in a pattern.

Global registers include ID register 509, read address register 614, control register 505 and registers in register array 507. Additional global registers, active register 706, match register 708 and select register (not shown) may be used to designate PU 500 as active, matched, or selected for writing configuration data. The ID of a PU 500 is an ID that is unique across a chip containing multiple PUs and is used to identify what pattern has been detected in a data stream being coupled in parallel to more than one PU 500. The counter 714 is used to index through the stored pattern 601 for comparison to bytes 801 in an input data stream (from input bus 503) and the comparator (not shown) in compare unit 511 compares the pattern 601with the input data 801 one byte at a time.

When PU 500 comes online, all registers are initialized to zero (reset). Next PU 500 receives unique ID from the input bus 503 which is stored in ID register 509. PU 500 then waits until it receives additional commands. The first command is a select command which activates PU 500 to receive further configuration commands that apply to PU 500 only. At this point the global registers may be loaded. Bytes of data are sent to the register array 507 which include the pattern data 601 and the corresponding Opcode data 602. When the configuration is complete and the active register 706 is set to "active", PU 500 waits for the packet reset signal 802 to enable the read address 614. This indicates that a new input packet is being sent to the PU 500 to begin the matching phase.

During the matching phase, one byte is sent to PU 500 at each clock cycle. PU 500 compares the byte stored (601) in the current register array position (determined by the address 614) in register array 507 with the input byte in input register 504 and checks the Opcode (602) for the byte in the current register array position of the pattern stored in 601. If there is a match or the Opcode 602 is set to a single wild card match, the pointer is incremented to select the next read address in address register 614. If the Opcode 602 for the current byte in pattern 601 is set to multiple wildcard, the pointer to address register 614 holds its current value. If a match was not found, then the pointer is reloaded. This process continues until the pointer is at the last position of a pattern and a match occurs. At this point, the match register 708 is set in PU 500. The final phase of the process is to report the found match. If the match register 708 is set, the output logic circuitry 512 sends the ID of PU 500 to the output bus 513.

FIG. 1 is a block diagram of a parallel pattern matching engine (PPDE) 100 integrated circuit (IC) architecture. PPDE 100 provides multiple mode pattern matching and has a highly flexible, massively parallel architecture. PPDE 100 can perform exact, fuzzy, longest and maximum character pattern matching. Some of the possible applications that can benefit from the capabilities of PPDE 100's high

performance pattern matching are: network intrusion detection, database search image processing, lossless compression, and real-time data processing (sound, EKG, MRI, etc.). The architecture of PPDE 100 is highly flexible and scalable and may be adapted to specific applications.

PPDE 100 is an IC comprising multiple PU 500 units and other logic functions. Input/output (I/O) interface 101 couples PPDE chip 100 to system functions. I/O interface 101 couples 64 bits of input data to IC input bus 120 which in turn couples to input buffer 103. Data is written into input buffer 103 in locations determined by write address 102. Data is read from input buffer 103 using read address 108. Data is read from input buffer 103 in 8 bit bytes using multiplexer (MUX) 115 controlled by select line logic 109. Input bus 503 is coupled to each of the N PU 500 units. I/O interface 101 also couples control data to global control 107 which sends 24 bits of ID data on ID bus 501 and 4 bits of control data on control bus 502 to each PU 500 unit (PU1-PUn).

FIG. 9 is a flow diagram of method steps in pattern matching using a PU 500 suitable for practicing embodiments of the present invention. In step 901, a packet reset is received indicating that configurations of the PU 500 is complete and a new packet (input pattern) is being sent to the PU and it should begin the matching process. In step 902 a first pattern byte of the pattern is retrieved. In step 903, the first pattern byte is compared to the first byte in the input data stream and a test is done to determine if they compare. The first pattern byte is indicated by an address pointer (pointer). If there is a compare in step 903, then a test is done in step 910 to determine if Opcode 602 is set to "match" for the present pattern byte (in this first pass it is the first pattern byte). If the Opcode 602 is set to "match", then the pointer is incremented by one to move to the next pattern byte as this is a desired result. If Opcode 602 for the present pattern byte is not set to "match", then in step 911 Opcode 602 is tested to determine if it is set to "inverse". If Opcode 602 is set to "inverse", then this is not a desired result and the pointer is reloaded back to the first pattern

byte in step 913 if it is not already there. A branch is then taken back to step 902. If Opcode 602 is not set to "inverse" in step 911, then Opcode 602 is tested to determine if it is set to "last" indicating the pattern byte is the last byte in the pattern. If Opcode 602 is not set to "last" in step 912, then the pointer is incremented in step 914 and a branch is taken back to step 902. If Opcode 602 is set to "last" in step 912, then the pointer is "frozen" and a branch is taken back to step 901 awaiting a new packet reset to restart match processing.

If the pattern byte and the input data byte do not compare in step 903, then in step 904 a test is done to determine if Opcode 602 is set to "match" for the pattern byte. If Opcode 602 is set to "match" in step 904, then this is not a desired result and the pointer is reloaded back to the first pattern byte in step 913 if it is not already there. A branch is then taken back to step 902. If Opcode 602 is not set to "match" in step 904, then a test is done in step 905 to determine if Opcode 602 is set to "inverse". If Opcode 602 is set to "inverse" in step 905, then this is a desired result and the pointer is incremented in step 914 and a branch is taken back to step 902. If Opcode 602 is not set to "inverse" in step 905, then a test is done in step 906 to determine if Opcode 602 is set to "wildcard". If Opcode 602 is set to "wildcard" in step 906, then this is a desired result and the pointer is incremented in step 914 and a branch is taken back to step 902. If Opcode 602 is not set to "wildcard" in step 906, then a test is done in step 907 to determine if Opcode 602 is set to "multiple wildcard". If Opcode 602 is set to "multiple wildcard" in step 907, then the pointer is held in step 908 and a branch is taken back to step 902. If Opcode 602 is not set to "multiple wildcard" in step 907, then in step 909 the pointer is reloaded and a branch is taken back to step 902.

The operations discussed relative to FIG. 9 are called regular expression matching. These regular expressions are used within matching modes used by the PPDE incorporating multiple PU 500 units according to embodiments of the present invention.

FIGS. 11A-11F actions taken relative to a pattern 601 when comparing to an input data stream 750. FIG. 11A illustrates three clock cycles of the case 1100 where input data 750 is "AAC" being compared to pattern data 601 as "ABC" where each pattern byte has an Opcode 602. The actions 1101 are taken in response to the Opcodes 602. In clock cycle 1, pointer 614 starts at the byte ("A") in pattern 601. The first byte of input data 750 is also an "A". Opcode 602 for the first byte in pattern 601 is set to "match". Since the first byte of input data 750 and pattern 601 compare and Opcode 601 is set to "match", the pointer is incremented moving to the second byte in pattern 601 which is a "B". This happens in one clock cycle, therefore, in the second clock cycle (labeled 1102 because it is significant to the particular pattern in FIG. 11A), the second byte in input pattern 750 ("A") is compared to the second byte in pattern 601 ("B"). The Opcode 602 for the second byte of pattern 602 is set to "match". Since these two bytes do not compare, the sequence "AB" in pattern 601 cannot match the first two bytes "AA" of input data 750 as required by the Opcode 602. Therefore, in clock cycle 2 (1102), pointer 614 is reloaded with the address of the first byte in pattern 602 and comparison begins again. In clock cycle 3, the third byte in input data 750 is compared to the first "A" in pattern 602.

FIG. 11B illustrates the case 1110 where the bytes sequence of input data stream 750 as "CDE" does match pattern 602 as a "CDE" but an Opcode 602 on one of the pattern bytes is set to "inverse" indicating that a match between a byte in input data 750 and a byte in pattern 601 is not desired. In clock cycle 1, the first "C" in input data 750 matches the "C" in pattern 601 and the Opcode 602 is set to "match". Since this is a desired result the pointer 614 is incremented and the second byte ("D") of input data 750 is compared to the second byte ("D") of pattern 601 and these bytes do compare. However, the Opcode 602 is set to "inverse" and a match is not desired, therefore in clock cycle 2 (1103) the pointer 614 is reloaded and the first byte of pattern 601 is again selected. In clock cycle 3, the third byte "E" in input data 750 is compared to the first byte "C" of pattern 601. The example of FIG. 11B is "looking"

for an input sequence "C!DE" where the "!D" indicates any character but not "D" is acceptable.

FIG. 11C illustrates case 1120 where a complete pattern 601 is shown with an Opcode 602 set to "last". In clock cycle 1, the first byte "F" in input data 750 matches with the first byte "F" in pattern 601 and Opcode 602 is set to "match". Since this is a correct result, pointer 614 is incremented. In clock cycle 2, the second byte "G" in input data 750 matches with the second byte "G" in pattern 601 and Opcode 602 is set to "match". Again, pointer 614 is incremented as this is a correct result. In clock cycle 3 (1104), the third byte "H" in input data 750 matches the third byte "H" in pattern 601. In this case, Opcode 602 is set to "last" indicating that the third byte is the last byte in a complete pattern 601 (in this case "FGH"). In this case the pattern "FGH" is detected in input data 750 and a match signal can be assert.

Since there is additional input data 750, pointer 614 is reloaded back to the first byte in pattern 601 and the matching process continues "looking" for additional occurrences of the complete pattern "FGH" in succeeding bytes of input data 750.

FIG. 11D illustrates case 1140 where a pattern 601 byte has Opcode 602 set to "inverse" and the bytes do not compare. In clock cycle 1, the first byte "I" in input data 750 matches the first byte "I" in pattern 601 and the Opcode 602 is set to "match". Since this is a desired result, the pointer 614 is incremented and the second byte ("J") of input data 750 is compared to the second byte ("I") of pattern 601 and these bytes do not compare. However, the Opcode 602 is set to "inverse" and no match is a desired result; therefore, in clock cycle 2 (1105), the pointer 614 is incremented and the third byte "K" of pattern 601 is again selected. In clock cycle 3, the third byte "K" in input data 750 is compared to the third byte "K" of pattern 601. Again, a match is detected and the pointer 614 is incremented. The example of FIG. 11D is "looking" for an input sequence "I!JK" where the "!J" indicates any character but "J" is acceptable.

FIG. 11E illustrates case 1130 where pattern 601 matches a sequence in input data 750 and the Opcodes 602 are set to "match". In clock cycle 1, pointer 614 starts at the byte ("L") in pattern 601. The first byte of input data 750 is also an "L". Opcode 602 for the first byte in pattern 601 is set to "match". Since the first byte of input data 750 and pattern 601 compare and Opcode 601 is set to "match", the pointer 614 is incremented to the second byte in pattern 601 which is an "M". In the second clock cycle, the second byte in input pattern 750 ("M") is compared to the second byte in pattern 601 ("M"). The Opcode 602 for the second byte "M" of pattern 602 is set to "match". Since these two bytes compare, the pointer 614 is again incremented. In clock cycle 3, the third byte "N" in input data 750 is compared to the third byte "M" in pattern 602. Since they compare, the pointer is again incremented. FIG. 11E illustrates a partial match of "LMN" in pattern 601 to the sequence "LMN" in input data 750.

FIG. 11F illustrates case 1150 where there is NOT a pattern match and the wildcard Opcode is set for a byte in the pattern 601. In clock cycle 1, the "O" in input data 750 matches with the "O" in pattern 601. Since the Opcode 602 is set to "match", the pointer 614 is incremented. In clock cycle 2, second byte "O" of pattern 601 does not match the "P" in the second byte of input data 750. However, since Opcode 602 is set to "wildcard" any character is accepted and pointer 614 is again incremented. In clock cycle 3, the third byte "Q" of pattern 601 matches the third byte "Q" in input 750 and pointer 614 is incremented. In this case, the sequence "O·Q" is found where "·" indicates any character.

FIG. 11G illustrates case 1160 where there is not a pattern match and a byte of pattern 601 has the Opcode 602 set to "multiple wildcard" (shown as simply "multiple"). In clock cycle 1, the first byte "T" in pattern 601 does not match the first byte "R" in input data 750. However, since Opcode 602 is set to "multiple", the pointer 614 is held at its present position (in this case, first byte of pattern 601). In clock cycle 2, the first byte "T" of pattern 601 does not compare with the second byte

in input data 750. Since Opcode 602 remains set to "multiple", the pointer 614 is held at the first byte of pattern 601. In clock cycle 3, the first byte "T" of pattern 601 does compare with the third byte of input data 750 and pointer 614 is incremented to the second byte of pattern 601. In clock cycle 4, the second byte of pattern 601 does compare with the fourth byte of input data 750 and the pointer 614 is again incremented. In clock cycle 5 (not shown), the third byte of pattern 601 matches the fifth byte in input data 750 and the pattern "TUV" is detected in input data 750.

The PPDE 100 has four matching modes: exact, longest, maximum and fuzzy. Exact matching may be used for aligned or non-aligned data and may incorporate the regular expressions such as single wildcard, multiple wildcard, inverse, or inclusive set. The exact matching mode may be utilized in applications such as network intrusion where line speed matching is critical and a binary match or not match response is only needed.

In the longest match mode, each PU 500 unit keeps track of the number of consecutive bytes matched and does not reset until the end of a pattern packet. In the longest match mode, each PU 500 outputs the number of matched bytes along with its ID to the ID selection unit 114 (FIG. 1A). ID selection unit 114 then outputs the ID of the PU 500 with the maximum number of matched bytes along with the length value of the longest match to the output buffer 105.

In the maximum matching mode, each PU 500 keeps track of the number of bytes matched and does not reset until the end of a pattern packet. In this mode, each PU 500 outputs the number of matched characters along with its ID to the ID selection unit 114. The ID selection unit 114 then outputs the ID of the PU 500 with the maximum number of matches and the value of the maximum number to the output buffer 105.

In the fuzzy matching mode, each PU 500 "looks" for the closed pattern and then outputs the ID of the PU 500 with the closest match and a corresponding

distance value quantifying the closeness of the match to ID selection unit 114 which in turn outputs the results to the output buffer 105. The distance is the result of a comparison between the input Pattern and the Reference pattern (RP) previously stored in memory. The distance calculation method is based on a norm that is user selectable. Several norm can be used , the norm can uses the "absolute value of a difference" operator. The successive elementary distances can be summed in the case of the Manhattan distance , i.e. dist = sum (abs (IEi-REi)) or the maximum value thereof is selected in the case of the maximum norm to determine the final distance. i.e. dist = max (abs (IEi-REi)) where IEi (Input Element) and REi (Reference Element) are the components of rank i (variable i varies from 1 to k) for the input pattern IP and the stored prototype Reference pattern RP respectively. Note that "abs" is an usual abbreviation for "absolute value". Other norms exist, for instance the L2 norm such as dist = square root (sum $(IEi-REi)^2$. The L2 norm is said to be "Euclidean" while the Manhattan and maximum norms are examples of "non-Euclidean" norms. Other Euclidean or non-Euclidean norms (such as the match/no match) are known for those skilled in the art. In particular, the "match/no match" norm, represented by the "match (IEi, REi)" operator is extensively used. The closest match is the pattern with the lowest result. Fuzzy matching is useful in image processing and real time data processing where the input data stream may have white noise superimposed on data.

FIG. 2A illustrates an example of the exact matching mode 200 using a PPDE 100 according to embodiments of the present invention. Patterns 203 correspond to ID numbers 205 numbered *1-n* and identify *n* PU 500 units incorporated into a PPDE 100.. Input pattern 201 would be sent in parallel to each of the *n* PU 500 units. In this mode, PPDE 100 is programmed to find if any of the *n* patterns are found in input data stream 201. By inspection, one can see that only pattern "4" is found in its exact sequence in the portion of input data stream 201 shown. In this case, the ID of the PU 500 with the exact match (in this case, "4" is the ID) would be outputted (output 204)

to ID selection unit 114 (not shown) which would send the value to output buffer 105 (not shown).

FIG. 2B illustrates an example of the longest match mode 220 using a PPDE 100 according to embodiments of the present invention. Again, input data stream 201 is coupled in parallel to *n* PU 500 units with ID numbers 205 numbered *1-n*. In this mode, PPDE 100 is programmed to determine the most consecutive bytes in the patterns 213 that appear in input data stream 201. Again, by inspection one can see that pattern "4" has the longest match with 5 consecutive bytes "ABCDE" appearing in the input data stream 201. In this case, the ID of the PU 500 with the longest match (in this case, "4" is the ID) would be outputted (output 204) along with the longest match value of "5" (output 206) to ID selection unit 114 (not shown) which would send the value to output buffer 105 (not shown).

FIG. 2C illustrates an example of the maximum match mode 230 using a PPDE 100 according to embodiments of the present invention. Again input data stream 212 is coupled in parallel to *n* PU 500 units with ID numbers 205 numbered *1-n*. In this mode PPDE 100 is programmed to determine the maximum number of bytes in the patterns 223 that appear in input data stream 212 not necessarily in consecutive order. Again, by inspection one can see that pattern "4" has the maximum number with 5 matching bytes "ACYEF" appearing in the input data stream 212. In this case, the ID of the PU 500 with the maximum number of matches (in this case, "4" is the ID) (output 204) along with the maximum number value of "5" (output 206) are outputted to ID selection unit 114 (not shown) which would send the value to output buffer 105 (not shown).

FIG. 2D illustrates an example of the fuzzy match mode 240 using a PPDE 100 according to embodiments of the present invention. Input data stream 222 is coupled in parallel to *n* PU 500 units with ID numbers 205 numbered *1-n*. In this example, input data stream 222 is an analog signal which would be digitized and each 8 bit input value would be sent to the *n* PU 500 units in parallel. In this mode, PPDE

100 is programmed to determine which of the patterns 233 most closely matches input data stream 222. Again, by inspection one can see that pattern "4" has the closest match. In actual operation, distance circuitry 611 (not shown) would be used to make this determination. In this case, the ID of the PU 500 with the closest match (in this case, "4" is the ID) (output 204) along with the distance value of "10" (output 206) would be outputted to ID selection unit 114 (not shown) which would send the value to output buffer 105 (not shown).

FIG. 3 is a block diagram illustrating the scalability of PPDE 100. The architecture of PPDE 100 allows for multiple chips to be cascaded. This feature may be used to either increase the number of processing units or to increase the performance by splitting the input data amongst the several chips. FIG. 3 illustrates the direct correlation between the number of chips and the number of PU 500 units. Block 303 shows the standard performance of one PPDE 100 chip. As PPDE 100 chips are added (by cascading) along the X axis the performance increases. Also, as the number of PU 500 units per PPDE 100 chip are added along the Y axis, the performance increases. Block 301 illustrates that by adding 4 chips (1500 PU 500 units) processing is increased to 8 Gb/sec for 1500 patterns. Block 304 illustrates using 4 chips to increase the number of patterns while maintaining the processing speed of 2 Gb/sec. Block 302 illustrates adding 5 groups of 4 chips coupled to process 6000 patterns to allow a system that can process 6000 patterns at 10 Gb/sec.

FIG. 4 illustrates a performance table of a PPDE 100 chip. Using a .13 micron CMOS technology, a PPDE 100 with 1500 PU 500 units would result in an 8 millimeter (mm) by 8 mm chip dimension. This corresponding PPDE 100 would achieve a bandwidth of 2 Gb/sec with a 250 MHz clock frequency wherein an 8 bit byte is processed each clock cycle. At this speed, the PPDE 100 chip would dissipate about 300 milliwatts (mw) of power and would compute at 1.25 tera operations per second. The PPDE 100 has the capability to be set in a standby mode in which it would consume a minimal amount of power. A PPDE 100 may be used with any I/O

interface 101. Using a Peripheral Component Interconnect (PCI) protocol, a maximum of 4 Gb/sec may be received at I/O 101. The PCI connection would have 88 I/O signals, 64 of which would be incoming data with 24 reserved for control.

A network processor (NP) is a programmable CPU chip that is optimized for networking and communications functions. It offers network equipment vendors an off-the-shelf alternative for building routers, switches and access devices much faster than by designing a custom ASIC chip. The network processor is programmed to perform the packet processing supported by the device and is expected to be widely used in all but the highest-end products.

FIG. 10 is a block diagram of a network processor 1000 suitable for practicing embodiments of the present invention. Physical layer devices 1007 have direct memory unit (DMU) busses 1015 and 1014 coupled to intermediate data storage PMM-UP 1009 and PMM-DN 1008, respectively. In this way, data is received from and transmitted to physical layer devices 1007. EPC 1006 couples to both Enqueue/Dequeue scheduler (EDS)-UP 1005 and EDS-DN 1004 units via connections 1050 and 1056, respectively (UP and DN represent directions of data flow). EPC 1006 also has internal on chip storage 1011. Selected units in EPC 1006 use an interface connection 1054 to the outside world (e.g., internet connection). Network processor 1000 may also be coupled with external chip memory 1001 comprising SRAM and DRAM, as well as Data storage 1010. Network processor 1000 may also include this own memory, in this case the memory block 1001 is localize inside the Network processor. Data is coupled to EDS-UP 1005 via Packet Memory Module (PMM)-UP 1009 and interface unit 1012 which couples to switch fabric interface 1002. Interface unit 1012 comprises serial data memory (SDM), a serial interface (SIF), and a data align serial link (DASL), which is an IBM proprietary interface. Other interfaces than DASL 1012 may be used and still be within the scope of the present invention. Data comes from the switch fabric interface 1003 via interface unit 1013, which also comprises DASL, SIF and

SDM-DN. The data is then coupled to EDS-DN 1004 which in turn couples to PMM-DN 1008. In this manner, data enters and is put in the desired packet protocol by EPC 1006 and outputted to a physical layer device or back to the switch fabric. EPC 1006 comprises special purpose processors (Pico Processors) which are programmable by a manufacturer but not the end-user. In this way, a manufacturer may customize the network processor 1000 to their desired application without having to resort to making an ASIC. The processor engines in the EPC are sufficiently fast that the operation competes with a custom ASIC.

A PPDE 100 illustrated in FIG. 1 is suitable for practicing embodiments of the present invention and when incorporated with an NP may be configured to provide the first cost effective way of meeting network speed requirements with sufficient performance and scalability to match intrusion detection requirements. A PPDE 100 is designed for recognition and classification applications that currently require super-computer processing power. NP 1000 is designed for reading and writing data from and to the network and performing a quick classification and routing function.

A PPDE 100 is an adaptive and highly flexible architecture in comparison to existing IDS pattern matching hardware technologies. A PPDE 100 also does not have the speed limitations imposed by a general purpose processor because it calculates all the matches of an intrusion signature in parallel using its PU 500 units. Also, the adaptive capabilities of a PPDE 100 are required in intrusion detection because new attacks are being constantly discovered and the list of intrusion signatures continues to grow with no end in sight.

PPDE 100 units provide a flexible and massively parallel architecture as well as having virtually unlimited scalability. The PPDE 100 architecture is able to process at 1 Gb/sec or more. A PPDE 100 is an ideal solution for intrusion detection applications.

FIG. 12 is a block diagram of an IDS 1200 according to embodiments of the present invention utilizing an NP (e.g., NP 1000) and one PPDE or a plurality of PPDEs. Network input data 1201 is received from network connection 1003 in network processor 1000. Memory 1001 may be used to store programs, intruder signatures, identification (ID) data for each PU 500 in each of PPDE 100's 1207-1210. PPDE input bus 1206 is used to couple control data, intruder signature data, and ID data to each of the PU 500 units. Once the PU 500 units have been loaded with ID data and their intruder signature data, they are ready to be enabled to start comparing input data bytes to intruder signature data on a byte basis. Network input data 1201 is received by network processor 1000 and is analyzed to determine if the packets are valid. Invalid packets are discarded and valid packets are transformed into a format compatible with PPDE 1207-PPDE 1209. Valid packets are also forwarded with routing data as output data 1208 on network connection 1002. The PU 500 units within the PPDE units 1207-1209 communicate via a cascade bus 1210 which allows exchange of information necessary when PU 500 units are cascaded. Circuitry forming a cascade bus 1210 is detailed relative to FIG. 14 and FIG. 15. Cascade bus 1210 allows control data to dynamically configure the PU 500 units within PDPE 1207-1209 for intruder signatures longer than a single unit can handle or to expand the number of intruder signatures that may be processed in parallel.

If a PU 500 unit detects a match to a particular intrusion signature, then it outputs its ID data to network processor (NP) 1000 on I/O bus 1206. NP 1000 is then able to look up the detected intruder signature in memory 1001. Memory 1001 also contains any action code associated with the detected intruder signature. Network processor 1000 executes the action code to which may include reporting the detected intrusion along with countermeasure procedures to counter the affects of the detected intrusion. Since the network data is coupled in parallel to a large number of PU 500 units, the IDS system 1200 is able to compare input data at network communication speed. If new intruder signatures are identified, network processor 1000 receives the new signatures and writes them into memory 1001 via connection 1205. The new

intruder signatures are then loaded into unused PU 500 units along with corresponding ID data. Control data is coupled to PPDE 1207-1209 which configures the PU 500 units as necessary whether to cascade units for additional signatures or to link PU 500 units to match a lengthy intrusion signature to large for a single PU 500.

FIG. 13 is a flow diagram of method steps used in embodiments of the present invention. In step 1300, a test is done to determine if network processor 1000 is executing any recovery routines or code that prevents it form processing network data. If NP 1000 is busy, a wait is executed waiting for network processor 1000 to be ready. If no recovery routines are active, then in step 1301, NP 1000 receives any intruder signatures and applicable action code corresponding to the received intruder signatures. In step 1302, the received intruder signatures are stored in memory 1001 along with corresponding ID data and any action code. In step 1303, intruder signatures are sent to PPDE 1207-1209 on PPDE I/O bus 1206 along with corresponding control data to configure the PU 500 units in PPDE 1207-1209. In step 1304, PPDE 1207-1209 are enabled to start comparing stored intrusion signatures to forwarded input data from NP 1000 in parallel. In step 1305, NP 1000 receives network input data 1201. In step 1306, NP 1000 analyzes the data packets and selects those packets to route as network output data 1208. The same data is then transformed and forwarded to PPDE 1207-1209 in parallel in step 1307. In step 1308, the forwarded network input data is compared to the store intrusion signatures using PU 500 units in PPDE 1207-1209. In step 1309, a test is done to determine if any matches have occurred. If no matches have occurred, then a branch is taken back to step 1308 where comparison continues. If a match has occurred, then in step 1310 the ID of the PU 500 detecting the match is sent to NP 1000. In step 1311, NP 1000 executes any action code corresponding to the detected intrusion signature. In step 1312, a test is done to determine if network data processing may continue. If no, then in step 1313 a branch is taken back to step 1300 awaiting completion of any recovery process. If the result of the test in step 1312 is YES, then a branch is taken back to step 1301 where any new intruder signatures are received by NP 1000.

FIG. 14 is a diagram of cascading circuitry 1401 used for cascade bus 1210 illustrated in FIG. 12. Cascade circuitry 1401 is used for communication to enable cascading of multiple PU 500 units within a PPDE 100 used in an IDS 1200 according to embodiments of the present invention. Control logic 1402 receives data on line 1413 and sends data on line 1412 to enable bi-directional communication to PU 1400. Cascade circuitry 1401 has communication logic 1406 and function logic 1410 coupled to control logic 1402. Link Out 1404 receives data outputted from a preceding PU 500 (not shown) and Link Out 1414 outputs data from control logic 1402 to a following PU 500 (not shown) If communication between PU 1400 and a preceding PU 500 is enabled, then a logic one is written to Chain In register 1403 which in turn enables AND logic gate 1405. Data from Link Out 1404 is coupled through AND gate 1405 to the input of OR logic gate 1416 and then to Link Out 1414. Link Out 1414 couples either data from Link Out 1404 or data from line 1413 of control logic 1402. If communication with the preceding PU 500 on Link Out 1404 is not desired, then Chain In register 1403 loaded with a logic zero which disables AND gate 1405. Data from a preceding PU 500 is coupled to control logic 1402 via AND gate 1405 (Chain In is a logic one) to line 1418. OR gate 1416 couples the data to line 1420 which is the input of OR gate 1411. The output of OR gate 1411 then couples the data via line 1413 to control logic 1402. Likewise, data from a following PU 500 sends data via Link In 1417 via line 1415 to the input of OR gate 1411. Again, the output of OR gate 1411 couples the data via line 1413 to control logic 1402. If Chain Out 1409 is loaded with a logic one, data from a following PU 500 is coupled via AND gate 1408 to Link In 1407 which is coupled as the Link In signal to the preceding PU 500.

FIG. 15 is a block diagram of communication between a PU 500 and two adjacent PU 500 units illustrating further cascade bi-directional communication and isolation using cascade circuitry 1401. The details of the cascade circuitry in each PU 500 1510-1530 is detailed and described relative to FIG. 14. FIG. 15 illustrates bi-directional communication between PU 500 units 1510 and 1520. Although PU 500

unit 1530 is physically connected to PU 500 unit 1520, PU 500 unit 1520 does not receive or send data to PU 500 unit 1530 with the logic states of the Chain In and Chain Out register bits as shown.

Chain In register 1501 is set to a logic zero and Chain Out register 1502 is set to a logic zero. This isolates PU 500 unit 1510 from any PU 500 unit (not shown) that is physically coupled to the left. Chain In register 1503 and Chain Out register 1504 are set to a logic one which enables bi-directional communication between PU 500 unit 1510 and PU 500 unit 1520. Again, Chain In register 1505 and Chain Out register 1506 are set to logic zero which isolates PU 500 unit 1520 from any PU 500 1530 and any other PU 500 unit (not shown) coupled to the right of PU 500 unit 1530.

Although the present invention and its advantages have been described in detail, it should be understood that various changes, substitutions and alterations can be made herein without departing from the spirit and scope of the invention as defined by the appended claims.

Although the present invention and its advantages have been described in detail, it should be understood that various changes, substitutions and alterations can be made herein without departing from the spirit and scope of the invention as defined by the appended claims.